Figure 1

Architecture and Dataflow

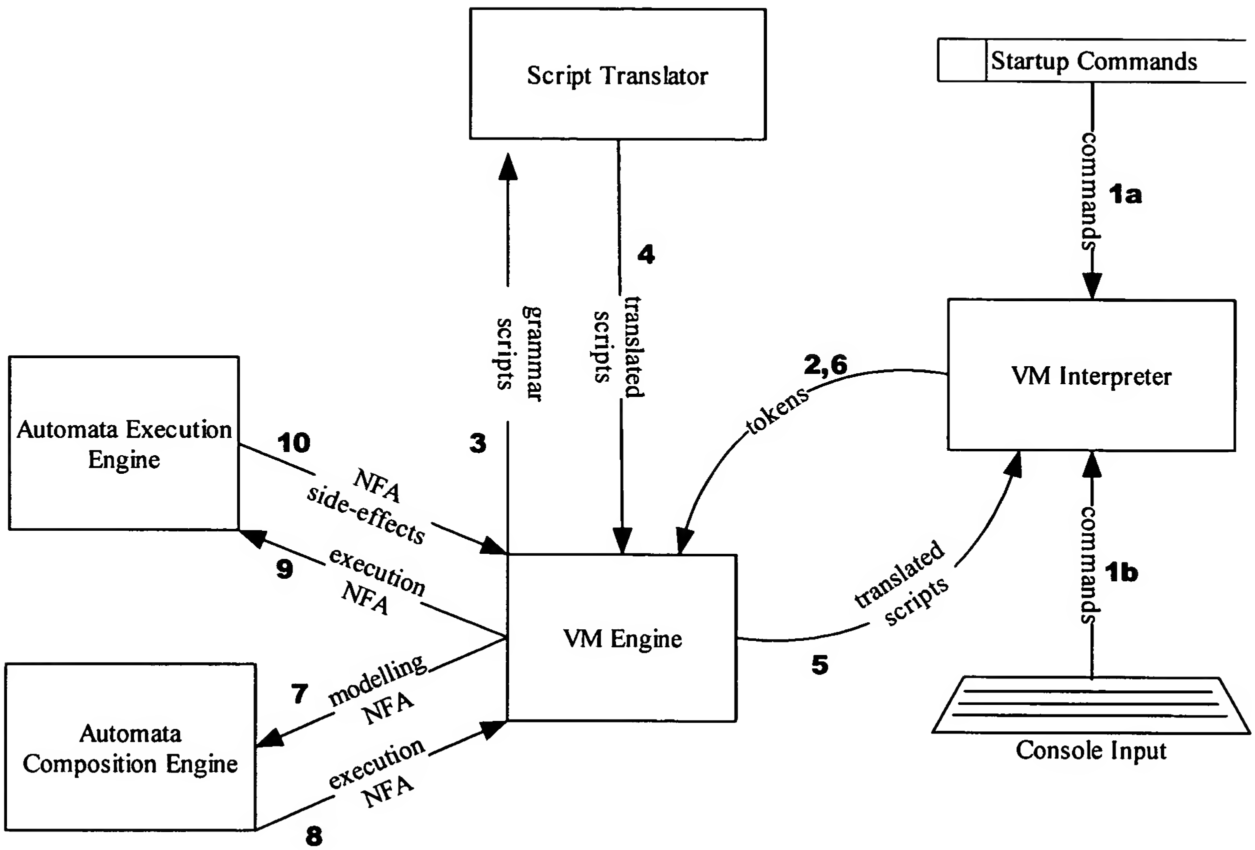

Figure 2

Name	Description
VM Interpreter	Converts FORTH words (Strings) to FORTH tokens and literals
VM Engine	Executes FORTH tokens and literals
Script Translator	Converts scripts written in "the grammar of this invention" to FORTH words
Automata Composition Engine	Composes modeling NFAs from FORTH regular expressions and converts NFAs to executable NFAs via subset-construction
Automata Execution Engine	Executes an NFA against the input stream, and returns side-effects as compiled FORTH tokens

Figure 3

VM Interpreter	**VM Engine**	**Script Translator**	**Automata Composition Engine**	**Automata Execution Engine**
EParser.java	Engine.java	Parser.java	AnyGrammar.java	AcceptState.java
Interpreter.java	EArray.java	Script.java	Arc.java	Automata.java
Console.java	EArrayRef.java	ScriptParser.java	ArcList.java	FastArc.java
Parser.java	EAtom.java	Stream.java	ArcNums.java	FastGraph.java
	EDict.java	Token.java	ButGrammar.java	FastNode.java
	EFrame.java	TokenStream.java	ButNotGrammar.java	IntQueueHeap.java
	EInstr.java	SUnit.java	CharClass.java	SimpleCharMap.java
	ELiteral.java	SUnit_BlockStmt.java	CharClassGrammar.java	
	EObjects.java	SUnit_BreakStmt.java	CharGrammar.java	
	EStack.java	SUnit_ContinueStmt.java	ConcatGrammar.java	
	EStream.java	SUnit_DeclStmt.java	ContainsGrammar.java	
	EType.java	SUnit_DoStmt.java	DoGrammar.java	
	OutStream.java	SUnit_EmptyStmt.java	Grammar.java	
	Stream.java	SUnit_Expr.java	Graph.java	
	Range.java	SUnit_ExprStmt.java	InhibitGrammar.java	
		SUnit_ForStmt.java	Instrs.java	
		SUnit_Frameable.java	IterateGrammar.java	
		SUnit_FuncDecl.java	Node.java	
		SUnit_IfStmt.java	NodeSet.java	
		SUnit_PrintStmt.java	NullGrammar.java	
		SUnit_ReturnStmt.java	OptionalGrammar.java	
		SUnit_RuleDecl2.java	RejectGrammar.java	
		SUnit_Stmt.java	RepeatGrammar.java	
		SUnit_StmtList.java	Segment.java	
		SUnit_TokenizeStmt.java	StringGrammar.java	
		SUnit_WhileStmt.java	SubsetNode.java	
		TypeDecl.java	TmplGrammar.java	
		VarDecl.java	Transition.java	
		VarDeclScope.java	TransitionArray.java	
		VarDeclStack.java	TransitionGroup.java	
			UnionGrammar.java	
			UpCount.java	

Figure 4

<table>
<tr><th>Operators</th><th>Purpose</th><th>Associativity</th></tr>
<tr><td>*</td><td>iterate and repeat</td><td>left-to-right</td></tr>
<tr><td>+   >+   <+</td><td>left-to-right and right-to-left concatenation</td><td>left-to-right</td></tr>
<tr><td>|</td><td>union</td><td>left-to-right</td></tr>
<tr><td>? :</td><td>conditional</td><td>right-to-left</td></tr>
<tr><td>but   butnot</td><td>subjunctive expressions</td><td>left-to-right</td></tr>
</table>

Figure 5

FORTH word	Stack operand(s)	Stack result(s)	Description
{xx	-	-	begins compilation of an executable instr-array for do-patterns
xx}	-	-	ends compilation of an executable instr-array for do-patterns
nop	-	-	the "nop" instruction
pdo	x-array-obj pat-obj x-array-obj	pat-obj	the low-level "do-pattern"
"AB"	-	"AB"	pushes the literal "AB" onto the FORTH stack
String.Pattern	str-obj	pat-obj	converts a FORTH string to a FORTH pattern object

Figure 6

FORTH word	**Stack operand(s)**	**Stack result(s)**	**Description**
swap	obj1 obj2	obj2 obj1	swaps top elements on FORTH stack
char.Pattern	char-obj	pat-obj	converts top element of FORTH stack from character object to Pattern object
CharClass.Pattern	charclass-obj	pat-obj	converts top element of FORTH stack from char-class to pattern
char.String	char-obj	str-obj	converts top element of FORTH stack from char to string
p?	pat-obj	pat-obj	modifies top Pattern on FORTH stack via unary "optional" operator
p|	pat-obj1 pat-obj2	pat-obj	creates a new pat-obj which is a union of the 2 patterns at top of stack
s+	str-obj1 str-obj2	str-obj	creates a new str-obj which is a concatenation of 2 strings at top of stack

Figure 7

FORTH word	Before stack	After stack	Action(s)
-> $0	-	var0-ref	Push reference to $0 local variable
-> '[AB]' CharClass.Pattern	-	pat-obj	Construct/push '[AB]' as CharClassGrammar object
-> assign	var0-ref pat-obj	-	Assign '[AB]' object to $0 local variable
-> $1	-	var1-ref	Push reference to $1 local variable
-> $0 valueof	-	var0-pat-obj	Push value of $0 local variable -> '[AB]'
-> assign	var1-ref var0-pat-obj	-	Assign '[AB]' object to $1 local variable
-> $0	-	var0-ref	Push reference to $0 local variable
-> $0 valueof	-	var0-pat-obj	Push value of $0 local variable -> '[AB]'
-> p?	pat-obj	pat-obj	Construct/push an OptionalGrammar object -> ?'[AB]'
-> $0 valueof	-	var0-pat-obj	Push value of $0 local variable -> '[AB]'
-> p?	pat-obj	pat-obj	Construct/push an OptionalGrammar object -> ?'[AB]'
-> p<+	pat-obj1 pat-obj2	pat-obj	Pop 2 grammar objects, construct a new ConcatGrammar object, and push -> ?'[AB]' <+ ?'[AB]'
-> swap	var0-ref pat-obj	pat-obj var0-ref	Swap the reference and grammar objects on stack
-> 1 get	pat-obj var0-ref	pat-obj var0-ref pat-obj	Push another reference to grammar object
-> assign	var0-ref pat-obj	-	Assign $0 to grammar obj -> ?'[AB]' <+ ?'[AB]'
-> drop	pat-obj	-	Drop top element
-> $2	-	var2-ref	Push a reference to $2 local variable
-> $0 valueof	-	var0-pat-obj	Get value of $0 variable -> ?'[AB]' <+ ?'[AB]'
-> $1 valueof	-	var1-pat-obj	Get value of $1 variable -> '[AB]'
-> p+	pat-obj1 pat-obj2	pat-obj	Pop top 2 grammar objs, construct new obj -> (?'[AB]' <+ ?'[AB]') + '[AB]'
-> assign	var2-ref pat-obj		Assign above grammar object to variable $2

Figure 8

Contemplated	Evolved to	Reason
['a', 'b', '0'..'9']	'[ab0-9]'	Given in section 8.3.5.1
anyother	butwithout butnot butalso	anyother cannot be modeled effectively as an NFA. The subjunctive form properly generalizes the negated character-class.
butwithout butnot butalso	but butnot	but and butnot are more accurately converses of each other; butwithout is just a design pattern based on butnot
righttoleft lefttoright	<+ >+	The <+ and >+ variations of the default + concatenation operator are more compatible with a C-style interpreter and C-style precedence/associativity rules.
P * N1* N2	P * N1..N2	Better to introduce a range type (and a binary range .. operator) and to overload a binary * operator that already exists in the C operator-list, than to introduce a new tertiary operator form.
reject accept	reject	accept cannot be modeled as an NFA (its meaning is context dependent); there are 2 possible interpretations to the meaning of accept; no use cases that demonstrate the ability to solve an otherwise unsolvable problem.
"unary after"	"unary before"	* ? + as unary operators must come before to be C expression syntax compatible
P1 otherwise P2 otherwise P3	-	This grammar has value but not in this form – will be part of a continuation patent.
choice<index>(Pattern[])	-	The "dynamic switch" can be modeled by line-by-line composition, leveraging a feature of instantiated production rules -- see example tmpl6.tok of section 8.4.2.4
concat<Pattern> (Pattern)	-	postponed
ignorecase<boolean> (Pattern)	-	not novel, postponed
"back-referencing" feature		postponed
"recursion" support		Postponed because of complications surrounding the subjunctive, which is implemented through subset construction.